Title: Doing Requirements Right the First Time!
Presenters: Theodore Hammer, Lenore Huffman, Dr. Linda H. Rosenberg, William Wilson, and Lawrence E. Hyatt
Track: Track 6 - Software Engineering
Day: Tuesday
Keywords: Requirements, Metrics, Quality, Testing

Abstract: The criticality of correct, complete, testable requirements is a fundamental tenet of software engineering. The success of a project, both functionally and financially, is directly affected by the quality of the requirements. Also critical is the complete requirements based testing of the final product. Modern tools for managing requirements allow new metrics to be used in support of both of these critical processes. Using these tools, potential problems with both the requirements and the test plan can be identified early. Problems include ambiguous or incomplete requirements, incomplete linkage of requirements to test cases, excessive or insufficient test cases. This paper addresses three critical aspects of requirements: definition, verification, and management. Project data collected from NASA Goddard Space Flight Center (GSFC) by the Software Assurance Technology Center (SATC) will be used to demonstrate these concepts and explain how any project, large or small, can apply this information.

Doing Requirements Right the First Time!

1. INTRODUCTION

It is generally accepted that requirements are the foundation upon which the entire system is built. And that requirement verification and validation is needed to assure that the functionality representing the requirements has indeed been delivered. However, all too often requirements are not satisfied, leading to a process of fixing what you can and accepting the fact that certain functionality will not be there. A better approach is to get the requirements right the first time, complete, concise and clear, that will provide the implementer a clear blue print with which to build the system. This is not done by magic but through the application of tools and metric analysis techniques in the areas of requirement specification, requirement verification and requirement management

Because both parties must understand requirements that the acquirer expects the provider to contractually satisfy, specifications are usually written in natural language. The use of natural language to prescribe complex, dynamic systems has at least two severe problems: ambiguity and inaccuracy. Many words and phrases have dual meanings that can be altered by the context in which they are used. Defining a large, multi-dimensional capability within the limitations imposed by the two dimensional structure of a document can obscure the relationships between

individual groups of requirements. The first part of this paper will look at terminology within NASA requirement specification that has led to ambiguity and potential misinterpretations.

Requirements based testing is critical in the implementation of software systems. Automated tools, if properly used, open the door to assessing the scope and potential effectiveness of the test program. Proper implementation of a database to not only track requirements at each level of decomposition, but also the tests associated with the verification of these requirements affords the project a wealth of information. From this database the project can gain important insight into the relationship between the test and requirements. The second part of this paper outlines some of the important insights into NASA project test programs developed from analyses of this type.

Requirements management is a volatile, dynamic process. The skill with which the project maintains, keeps current, tracks, and traces its set of requirements affects every phase of the project's software development life cycle – including maintenance. The ability to effectively manage requirements determines, months and/or years before project completion, how, when and how expensively completion will take place. Prior to processing a requirements, the schema for the requirement management database must be developed. The final portion of this paper describes some critical issues identified by the SATC that are needed to effectively manage requirements databases and discusses lessons learned on how to effectively design and maintain requirements databases.

This paper will demonstrate how metrics can help in these three areas of requirement development. Examples will be provided how metrics can identify areas of weakness that should be corrected, through the use of data from a large NASA project, Project X. Lessons learned will also be listed to aid in keeping a project, large or small, on track.

2. DEVELOPMENT ENVIRONMENT

In order to demonstrate how metrics can provide the insight needed to get the requirements right, data from a large NASA project, Project X, will be used. While the project must remain anonymous, a general understanding of the project's development environment is necessary. For clarity, we will also describe some development aspects that may not be standard in all environments. This project is implementing a large system in three main incremental builds.[1] The development of these builds is overlapping, design and coding of the second and third builds having been started prior to the completion of the first build. Each build adds new functionality to the previous build and satisfies a further set of requirements.

[1] Various names are used—deliveries, releases, builds—but the term *build* will be used in this paper.

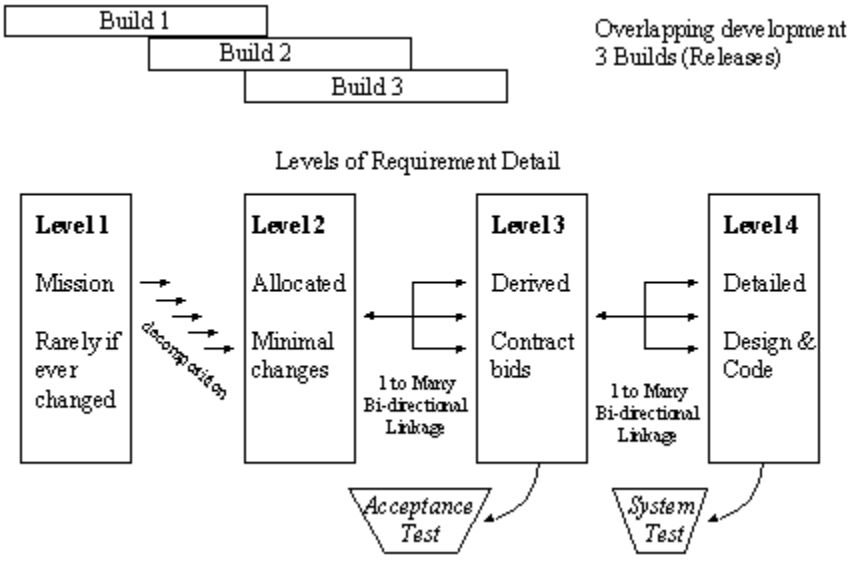


Figure 1 - Development Environment

The definition of requirements for this system started with the formulation of System Level Requirements, referred to as "Level 1" requirements. These are mission-level requirements for the spacecraft and ground system; they are at a very high level and rarely, if ever, change. We will not discuss requirements at this level because they are not stored in the requirements database under scrutiny. Level 1 requirements then undergo decomposition to produce Allocated Requirements, called "Level 2"; these requirements are also high-level and change should be minimal. Project's development started at this requirement level. Level 2 requirements are then divided into subsystems and a further level is derived in greater detail; hence, "Level 3 Derived Requirements." Generally, contracts are bid using this level of requirement detail. Each requirement in Level 2 traces to one or more requirements in Level 3. This is a bi-directional tracing, with Level 3 requirements refocusing into Level 2 requirements. The Detailed Requirements are found in "Level 4" requirements; these requirements are used to design and code the system. There is also a bi-directional tracing between Level 3 requirements and Level 4 requirements. To verify the requirement, two stages of testing are used. System Tests are designed to verify the Level 4 requirements and then Acceptance Tests are to be used to verify the Level 3 requirements.

3. REQUIREMENT SPECIFICATION

The importance of correctly documenting requirements has caused the software industry to produce a significant number of aids [1] to the creation and management of the requirements specification documents and individual specifications statements. However very few of these aids assist in evaluating the quality of the requirements document or the individual specification statements themselves. The SATC has developed a tool to parse requirements documents. The Automated Requirements Measurement (ARM) software was developed for scanning a file that contains the text of the requirements specification. During this scan process, it searches each

line of text for specific words and phrases. These search arguments (specific words and phrases) are indicated by the SATC's studies to be an indicator of the document's quality as a specification of requirements. ARM has been applied to 56 NASA requirement documents. Seven measures were developed, as shown below.

1. Lines of Text - Physical lines of text as a measure of size.

2. Imperatives - Words and phases that command that something must be done or provided. The number of imperatives is used as a base requirements count. [Shall, must or must not, is required to, are applicable, responsible for, will, should]

3. Continuances -Phrases that follow an imperative and introduce the specification of requirements at a lower level, for a supplemental requirement count. [As follows, below, following, in particular, listed, support]

4. Directives – References provided to figures, tables, or notes.

5. Weak Phrases - Clauses that are apt to cause uncertainty and leave room for multiple interpretations measure of ambiguity. [Adequate, as applicable, as appropriate, as a minimum, be able to, but not limited to, be capable of, effective, easy, effective, if effective, if practical, not limited to, normal, timely]

6. Incomplete – Statements within the document that have TBD (To be Determined) or TBS (To Be Supplied).

7. Options - Words that seem to give the developer latitude in satisfying the specifications but can be ambiguous. [Can, may, optionally]

It must be emphasized that the tool does not attempt to assess the correctness of the requirements specified. It assesses individual specification statements and the vocabulary used to state the requirements, and also has the capability to assess the structure of the requirements document.[2]

To see how this tool would be used to assess the "quality" of the requirements document, the Project X Level 3 requirements document was analyzed using the ARM Tool. Table 1 shows the results.

[2] This tool is available at no cost from the SATC web site http://satc.gsfc.nasa.gov

56 DOCUMENT	Lines of Text - Count of the physical lines of text	Imperatives - shall, must, will, should, is required to, are applicable, responsible for	Continuances - as follows, following, listed, in particular, support	Directives - figure, table, for example, note:	Weak Phrases - adequate, as applicable, as appropriate, as a minimum, be able to, be capable, easy, effective, not limited to, if practical	Incomplete (TBD, TBS)	Options - can, may, optionally
Minimum	143	25	15	0	0	0	0
Median	2,265	382	183	21	37	7	27
Average	4,772	682	423	49	70	25	63
Maximum	28,459	3,896	118	224	4	32	130
Stdev	759	156	99	12	21	20	39
Project X	34,664	1,176	714	873	13	480	187

Table 1 - Requirements Specification Analysis Example

Several things can be seen from this analysis. First, the document shows some strengths. There appears to be a good number of imperatives, and the number of weak phrases is low as compared to the family of NASA documents processed through the ARM tool to date. However, the document shows some significant weaknesses. The document has a large amount of text given the number of imperatives. This gives an indication of being a wordy document, which can have the effect of obscuring the requirements, preventing the requirements from being clear and concise. The document also has a large number of incomplete requirements, containing TBDs and TBSs. It could even be said that this document is not ready for use on this point alone, as this implies that there is still uncertainty about what the system is required to do. It is very difficult to build a system that has undefined requirements. Also this document has a large number of options, which increases the uncertainty about what is really required of the system that is to be developed. Options leave decisions about what the system is to do to the implementers, many times without sufficient direction or instruction about option selection criteria. As a result the implementation varies widely, anything from some of the options to none at all (especially since these items are options and not "really" required).

A further understanding of the requirements documentation can be achieved by looking at the document structure.

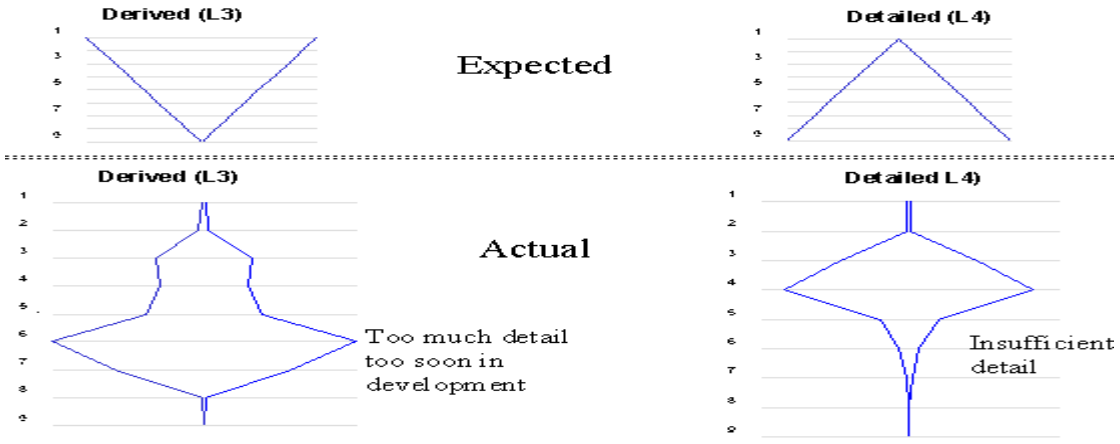


Figure 2 - Structure Level at Which Imperative Occurs

Figure 2 shows the expected structure, based on other NASA documentation, and actual structure for documentation from Project X. The expected structure is a graphical representation of the numbering structure used within the requirements documentation. The levels represent sub tiers within a section. For example four sub tiers would be 1.0, 1.1, 1.1.1, and 1.1.1.1. The expected graph for the Level 3 document indicates that there are many more high level requirements than detailed requirement expansions. This makes sense, as the Level 3 document is to define the overall requirements of the system and not provide details. The expected graph for the Level 4 document shows the opposite. There are many more detailed expansions of the requirements than of high level statements. Again this makes sense, as the detailed requirements document is to be the basis for the implementation of the system. The Project X documentation show some disturbing weaknesses. The Level 3 document shows a trend to over specify some of the requirements too early in the life cycle. The Level 4 document shows not enough detail. The weakness of the Level 4 document may be resultant from the trend to over specify requirements in the parent, Level 3, document, or most probably is the result of the Level 3 document having too many incomplete requirements and options (as seen from the first analysis using the ARM Tool).

Getting the requirements right in the specification has always been a desire of engineers but there has been little available in terms of analysis tools that would allow them to visualize the quality of the documentation. Now with the ARM Tool the quality aspects of the documentation can be visualized in such a way as to allow actions to be taken to improve the documentation.

4. REQUIREMENT VERIFICATION

Requirements testing is another important aspect of getting the requirements right. Though this may not be seen as directly to related to the issue of getting the requirements right, it is crucial because delivered capability cannot be determined without an effective verification program. In looking at the verification program, a further understanding of the nature of the requirements must be attained. This is done by looking at requirement stability and expansion. The linkage of requirements to test cases is reviewed, and then a test profile is made to characterize the entire test program. Again, data from Project X is used to demonstrate the utility of metrics in understanding requirement verification.

Requirement stability impacts the verification effort in that testing can not be planned or designed with the requirements continually in a state of flux.

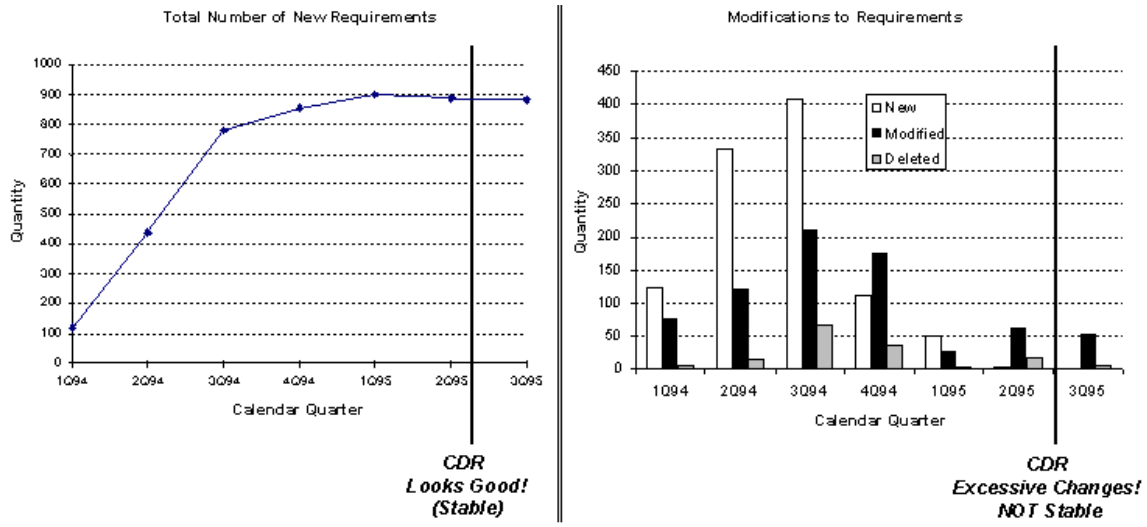


Figure 3 - Requirement Stabilization - Volatility

Figure 3 show how metrics can be used to gain insight into requirement stability and the importance of looking a particular issue in more than one way. This figure shows that the total number of requirements stabilized in time for the Critical Design Review (CDR), which is what is desired. However, when one looks at requirement stability in terms of new, modified, and deleted requirements one notices that the requirements are not that stable. There is almost constant change occurring in the modification of requirements. This will endanger the verification program. Another way of viewing requirement stability is to look at the allocation of requirements to the individual builds or releases. Figure 4 show the allocation of Level 4 requirements to Build 2 and Build 3 for Project X.

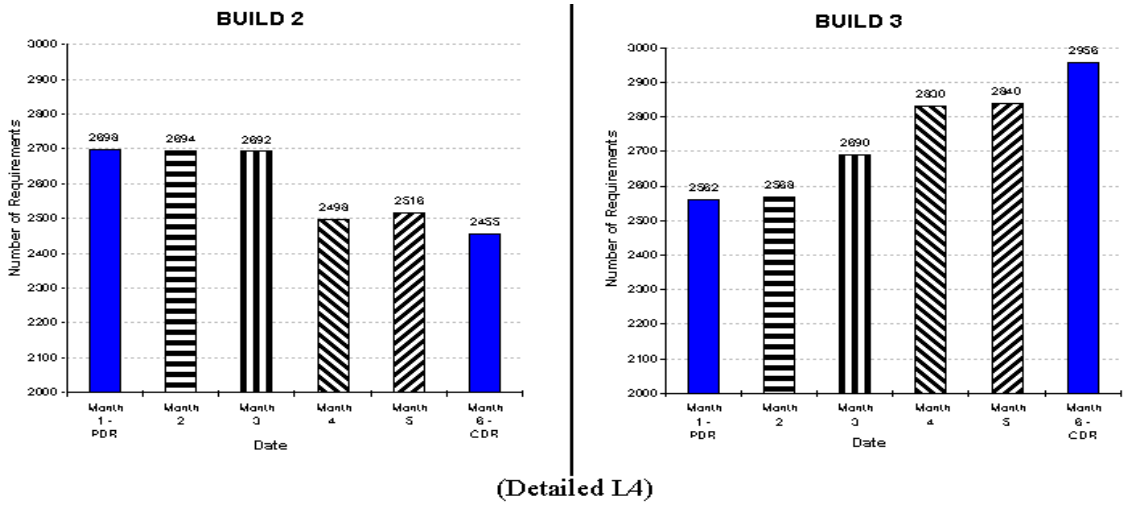


Figure 4 - Requirement Stabilization by Build

What can be seen is that requirements are continually being moved or reallocated from Build 2 to Build 3. This instability will make the implementation and verification of Build 3 difficult, as many requirements have been pushed into the last build in the development effort.

Requirement stability can be viewed in terms of requirement traceability and expansion. Requirements traceability is the linkage of the requirements at one level to the requirements at the next lower level. If there is missing linkage, a case can be made that possibly more requirements need to be written. Requirement expansion is the measure of how many requirements at the lower level, Level 4, were written to completely satisfy the Level 3 requirements. If there is little expansion in the number of requirements, a case may again be made again that there should be more requirements written to provide the level of detail necessary to implement the system. Figure 5 shows the linkage of Level 3 requirements to Level 4 requirements.

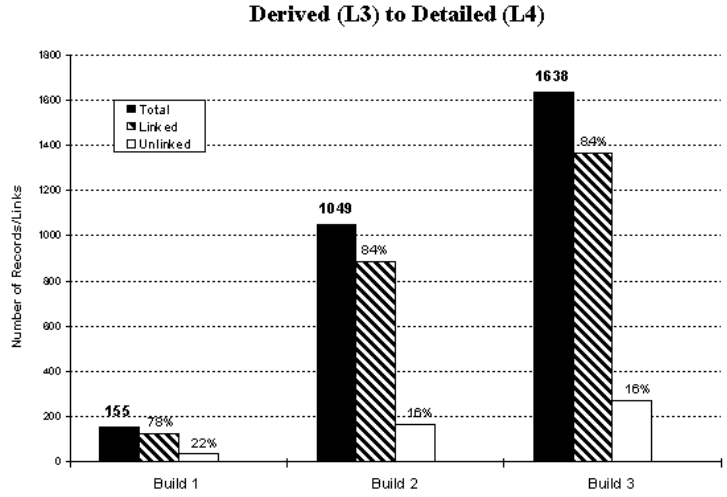


Figure 5 - Requirement Traceability

In all cases there is missing linkage (white bar of graph) between Level 3 and Level 4 requirements, indicating that the Level 4 requirements are potentially incomplete if a CDR was held for any one of these builds.

In reviewing requirement expansion, a comparison is made with data compiled from NASA projects which leads to an expected curve for requirement expansion that is bell shaped. This reflects that few requirements are expected to have little expansion or be expanded to a large number of requirements at the next lower level. As a result, there tends to be an average number of requirements written to decompose the Level 3 requirements to the next level of detail. Figure 6 shows the situation for Project X.

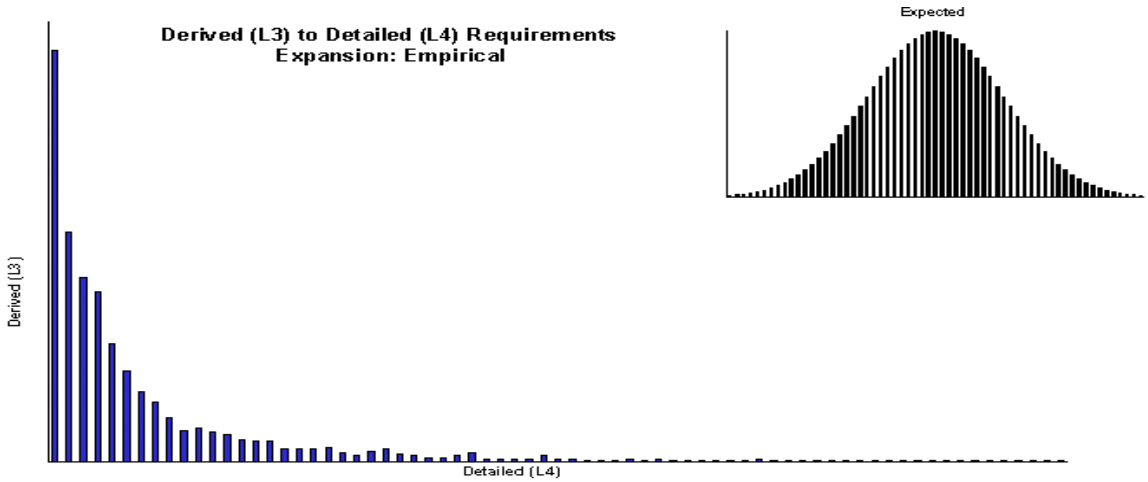


Figure 6 - Requirement Decomposition

Here we see that the Level 3 requirements for the most part have not been expanded while there are a few that have many requirements written to expand on the Level 3 requirements. This situation correlates very well with the metrics developed from the analysis of the documentation structure mentioned above, where the structure of the Detailed (L4) requirements specification showed a lack of detail. This lack of detail not only jeopardizes the implementation effort but also the development of effective verification procedures.

The objective of an effective verification program is to ensure that every requirement is tested, the implication being that if the system passes the test, the requirement's functionality in included in the delivered system [1,2]. An assessment of the traceability of the requirements to test cases is needed. It is expected that a requirement will be linked to a test case, and may well be linked to more that one test case as shown in Figure 7 [3,4].

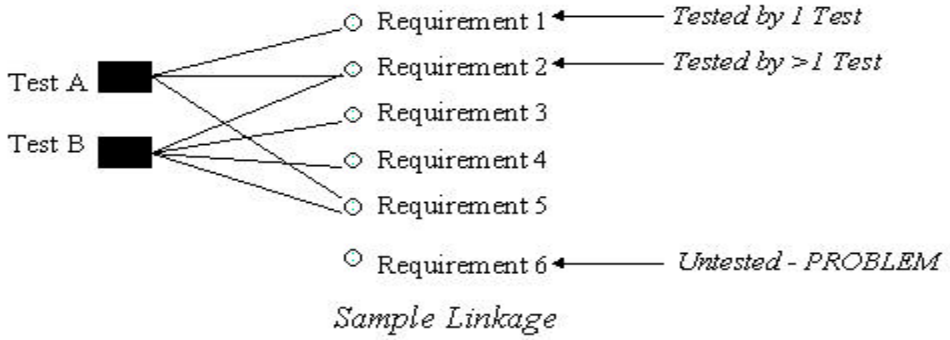


Figure 7 - Requirement Verification - Trace to Test Linkage

The important aspect of this analysis is to determine which requirements have not been linked to any test cases at all.

Figure 8 shows that the traceability of requirements to test cases for Project X around the CDR time frame for Build 2. The information was extracted from the requirements management database used in support of the development effort. The profiles show several problems.

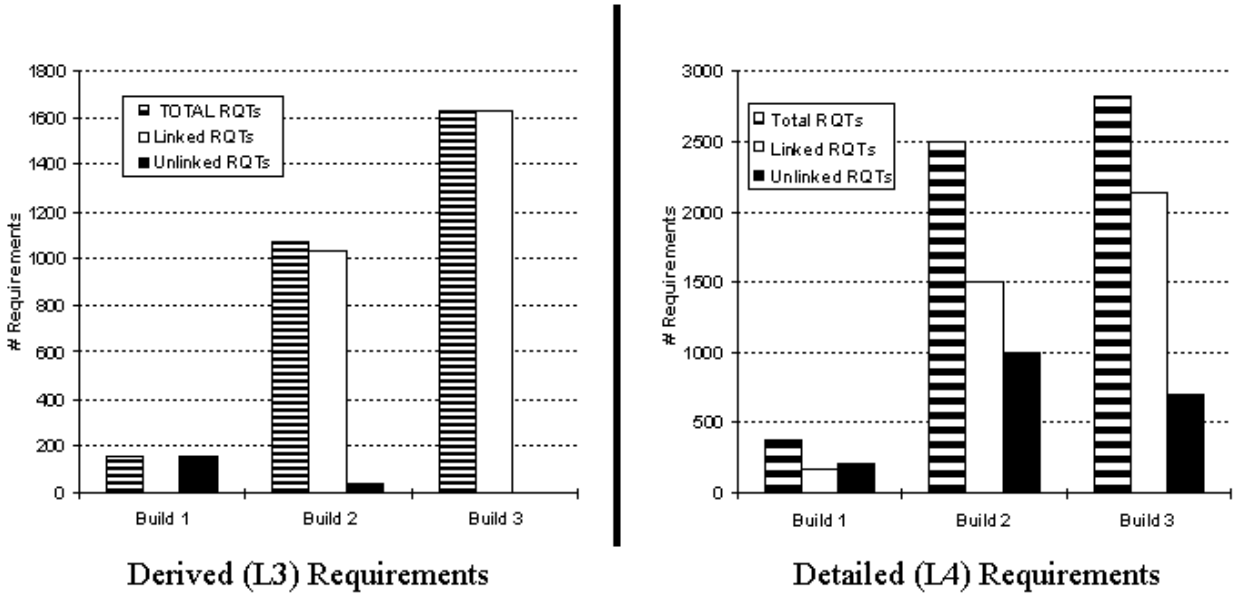


Figure 8 - Requirement Verification Trace to Test

First, the requirements management tool was not used effectively early in the project life cycle. This explains the poor traceability between the requirements and test cases for Build 1. Secondly, there seems to be a mix up in the test priorities by the implementer. The test program for Build 3 is further along than that for Build 2, when it is Build 2 that will be developed and tested before Build 3. Resources may have been inappropriately allocated to the development of the test program for Build 2. Lastly, the test program for the Level 4 requirements is behind that for the test program for the Level 3 requirements. Again, this is backwards. The first tests to be executed will be that for the Level 4 requirements, the system tests, and after that tests for the Level 3 requirements will be executed, the acceptance tests. An explanation for this problem may be found is a previously presented metric. Remember the metric showing the push of Level 4 requirement from Build 2 to Build 3. This movement of requirements from Build 2 to Build 3 may well be the cause of the lack of traceability of requirements to test cases. The test case developers may be having difficulty in keeping up with the changes in requirements resulting in a number of requirements in each build without a link to a test case.

Not only is it important to understand whether all the requirements are linked to test cases, but also to understand the character of the test program. This can be done by looking at the profile and relationship of requirements to test cases. This provides an understanding of the nature of the test program. Figure 9 shows an expected profile of unique requirements per test case based on data from NASA projects [5].

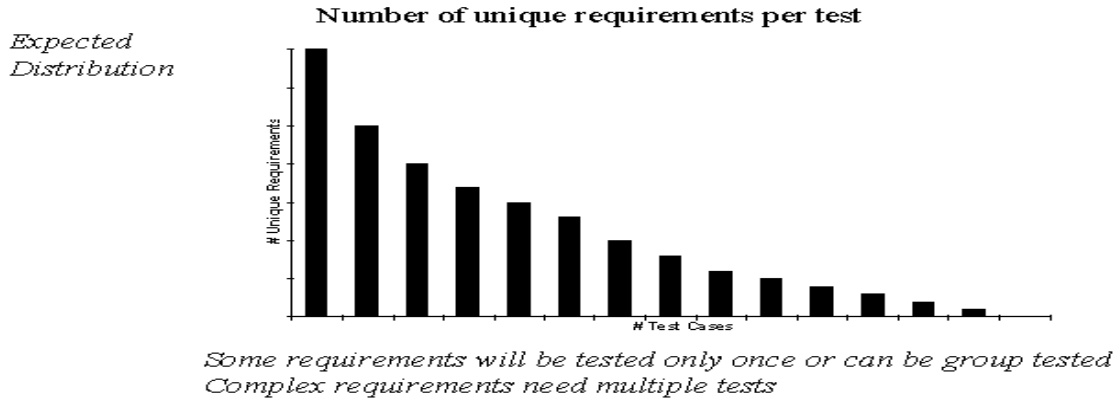


Figure 9 - Test Program Characterization Tests per Requirement

This profile shows that there is an expectation that there will be a large number of requirements tested by only one test case, and that there will be some number of requirements that will be tested by a multiple number of test cases. It is expected that the upper bound of multiple test cases will range in the tens. This makes sense, as more complicated requirements may require different test cases to thoroughly verify all aspects of the requirement. However, there is a limit on the number of test cases. As the number of test cases increases the difficulty in verifying the requirement increases, due to the complication in data analysis, understanding the results of the multiple tests cases, and understanding the impact of multiple test case results on the verification of the requirement. Figure 10 shows the requirement to test case profile for Project X. There is a good indication that there are a large number of requirements covered by just one test, making for a simple, easy to evaluate test program for a significant part of the system requirements. However, there are several instances for both Build 2 and 3 where the are several tests for unique requirements. Notice that for Build 2 that one requirement has been linked to 25 test cases, and in Build 3 that one requirement is linked to 51 test cases. This large number of test cases may well make it impossible to verify that these requirements have been implemented.

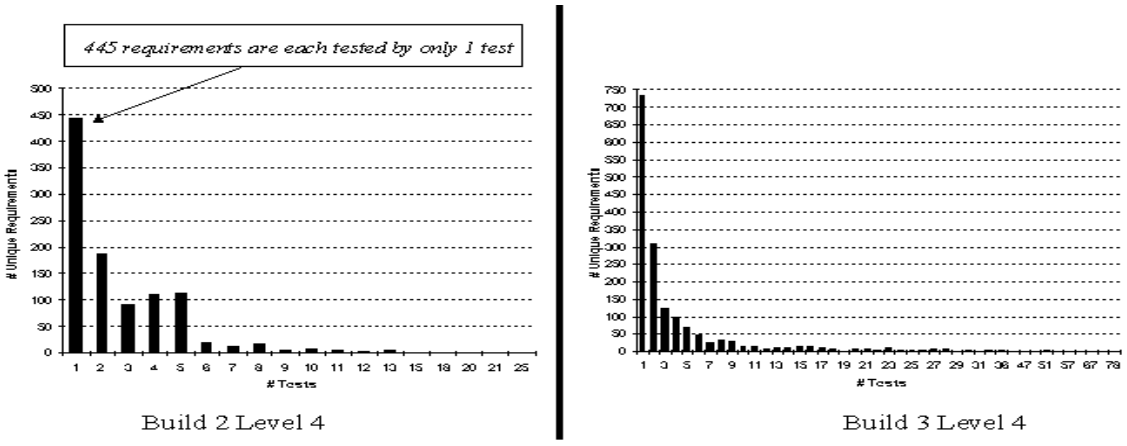


Figure 10 - Test Program Characterization Tests per Requirement

In summary the verification program for Project X has some strengths; the total number of new requirements is stable, and the Level 3 requirements have good linkage to tests for the acceptance test program. But there are also significant weaknesses. The data associated with Build 1 was not populated in the requirements management tool. There was a shifting of requirements between builds late in the requirement phase. Requirements were not completely decomposed from the Level 3 requirements to the Level 4. The Level 4 test program showed a significant number of requirements without links to tests. Test programs at both Level 3 and 4 showed some excessive testing of requirements.

5. REQUIREMENT MANAGEMENT

The use of tools to aid in the management of requirements has become an important aspect of system engineering and design. Considering the size and complexity of development efforts, the use of requirements management tools has become essential. The tools which requirement managers use for automating the requirements engineering process have reduced the drudgery in maintaining a project's requirement set and added the benefit of significant error reduction. Tools also provide capabilities far beyond those obtained from text-based maintenance and processing of requirements. Requirements management tools are sophisticated and complex – since the nature of the material for which they are responsible is finely detailed, time-sensitive, highly internally dependent, and can be continuously changing. Tools that simplify complex tasks require skill and a thorough understanding of their capabilities if they are to perform effectively over the lifetime of a project [6].

There are many requirement management tools to choose from. These range from simple word processors, to spreadsheets, to relational dbs, to tools designed specifically for the management of requirements such as DOORS (Quality Systems & Software - Mt. Arlington, NJ) or RTM Requirements Traceability Management (Integrated Chipware, Inc. - Reston, VA). The key to selecting the appropriate tool is the functionality (See Table 2 for a comparison of tool capabilities) provided and the capability to develop metrics from the data, secondary contained in the tool.

	Word Processor	Spreadsheet	Relational Database	Requirement Tool
Document config. mgt	X		X	X
Document preparation	X			X
Function decomposition			X	X
Report preparation			X	X
Requirement allocation		X	X	X
Requirement config. mgt		X	X	X
Requirement expansion			X	X
Requirement importation				X
Requirement simplification				X
Requirement storage	X	X	X	X
Requirement traceability			X	X
Test coverage/adequacy			X	X
Metrics			X	X

Table 2 - Requirement Repository Capabilities

The metric capability of the tool is important. It should be noted that most of the metrics presented in this paper to demonstrate how to do requirements the right way were developed from the data contained in a requirement management tool. Table 3 shows a comparison of the metric capability associated with the different tools. Clearly the relational database and requirements management tool provide the capabilities needed to effectively support the management of requirements.

	Word Processor	Spreadsheet	Relational Database	Requirement Tool
Document size	X			
Dynamic changes over time				X
Release size	X	X	X	X
Requirement expansion profile			X	X
Requirement types	X	X	X	X
Requirement verification			X	X
Requirement volatility	X	X	X	X
Test coverage			X	X
Test span			X	X
Test types	X	X	X	X

Table 3 - Requirement Repository metric Capabilities

The selection of a tool is only part of the equation. A thorough understanding of the tool capabilities and the management processes that will use the tool is necessary. The tool should not be plugged into the management processes with no thought as to the impact on the tool capabilities. Adjustments may be needed in the management processes and employment of the tool to bring about an efficient requirements management process. Without being aware of this issue the same problems may be encountered as by Project X. What follows is a discussion of Project X's experience in using a requirements measurement tool.

Project X’s focus in establishing a requirement management process was influenced by project organization. The project established a system management office for managing the high level requirements, a design group for managing the design requirements, integration groups for system testing, and acceptance test groups for the acceptance testing of the system (see Figure 11). With test identified within each test group, emphasis now shifts to testing-by-build. That is, instead of all IT (integration testing) and AT (acceptance testing) tests residing in their own class, tests were further subdivided by build - A, B, C - so that test classes now are labeled: ITA, ATA, ITB, ATB, ITC, ATC, etc. This again made sense at the time since one organization was responsible for one build and one type of test [6].

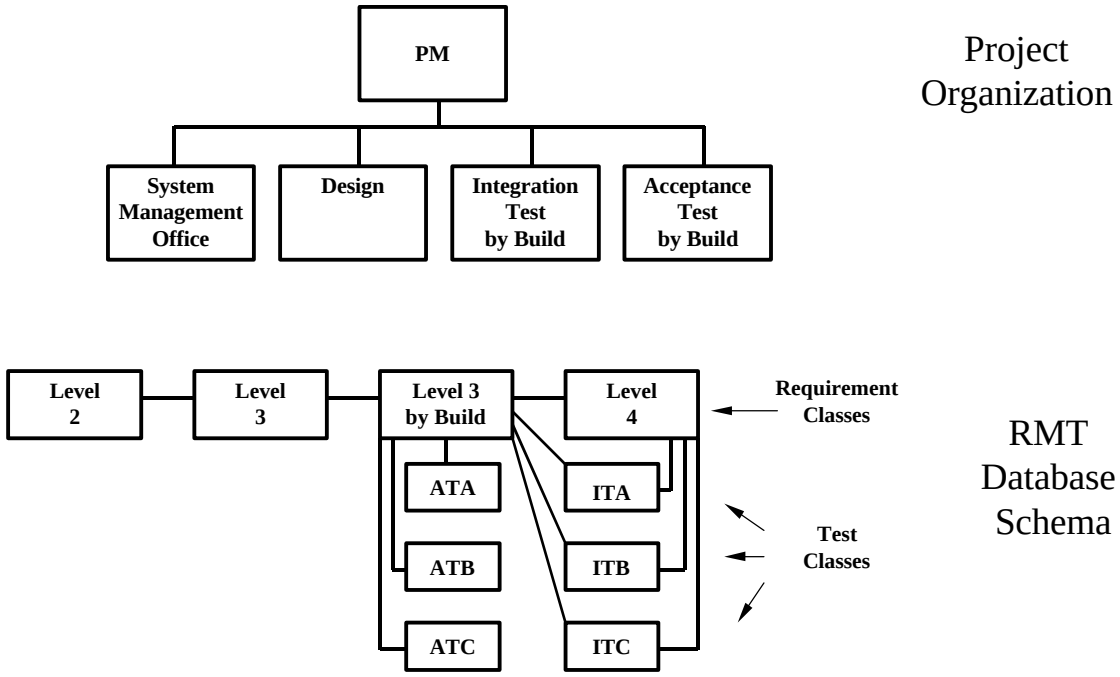


Figure 11 – Project Organization and Requirement Management Tool (RMT) Schema

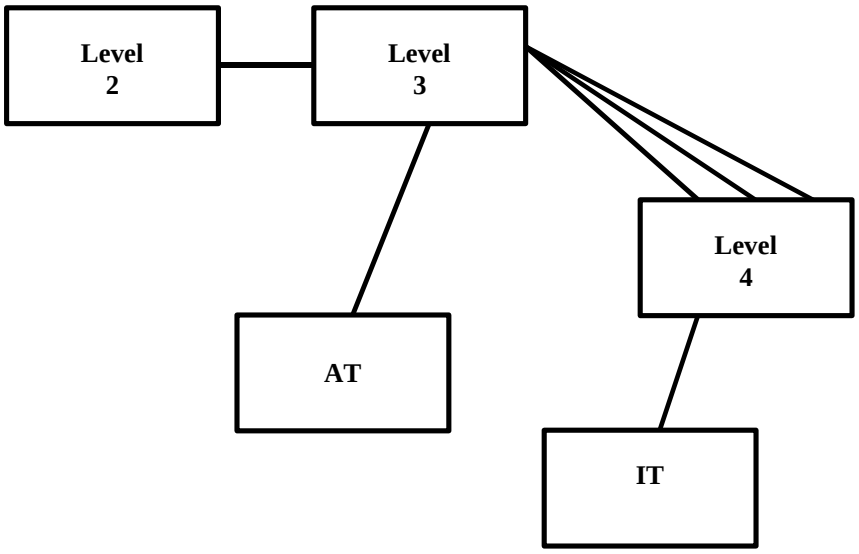
The established requirement management process was to have each group responsible for the data in its domain without consideration for how one data set related to another data set. As a result, the requirement management tool was set up to have a class for each of the organizational elements without regard to a.) how the requirements were being managed in each class and b.) whether the requirements were relating cleanly to requirements or tests of other classes. Figure 11 also shows the database concept that was developed for the project [6].

As can be seen there is a class for each of the organizational elements: Level 3 requirement class for the system management office, another Level 3 requirement class which simply assigns builds to each complex requirement, Level 4 requirements class for design engineering, and test classes by build for both integration and acceptance testing. (Note that for the Level 3 requirement class, an entire class is duplicated to another class simply for the purpose of assigning builds.) This produced an unnecessary or inaccurate compartmentalization of each set of requirements or test cases. [7] The 'expansion' traces between requirements, and between requirements and test cases were then established between the many classes. This seemed a reasonable approach, which provided an easily understood database schema, and a sense that each class was under the control of the appropriate organizational element [6].

The use of the tool in the manner selected by the project, while appearing reasonable on the surface was actually fraught with the flaw of inexperience, and ultimately worked against the clear management of project. Specifically, multiple classes mirrored organizational structure instead of a single class existing for each development phase. Figure 12 shows an example of a single class schema design. Though it appears similar to the Project X schema, the key is the use of a single class and simplified linkages. Project X appeared to have a simple solution but turned into a deadly mistake. This is much like using a screwdriver to open the car door. The surface

problem is solved, opening the door; but more serious problems have been introduced by that act. (E.g. broken lock).

A requirement in Class Level 3 with appropriate decomposition



Minimal Linkage

Figure 12 – Simplified Schema Design

With this multiple test class and requirement class approach, there was a natural tendency for the organizations to "improve" the data schema definitions assigned to them. Though there was a centralized configuration management office responsible for control of the database schema, each individual organization dictated to the CM Office so that there were many customers with many classes requesting many changes in isolation from each other; control was not effectively administered. This naturally led to losses in data integrity and prevented access to or use of important information about the requirements. Some information became specific to a particular organization that should have been available to all project organizations; other data became degraded and useless when it was no longer maintained [6].

Because multiple classes were implemented at the test-by-build level, fields were duplicated to each of the test classes; common information then became self-contained within each class. However, confusion developed between the test organizations as to which one was responsible for populating common data. The project started with a set of naming conventions, but these were corrupted as each organization customized its classes. Also, each organization interpreted the meaning of the common fields in different ways. In all of these cases it was not obvious to the configuration management office that there was a problem. The number of classes with similar fields helped to obscure a problem in entering data and worked against rigid compliance with a published data dictionary [7]. This all lead to inconsistent data entries and prevented effective data mining [8].

Another very significant problem that developed with the selected schema was the necessity of using a significant number of linkages to individual requirements between individual classes. This was necessary since each level of requirement was populated to a separate class. No class manipulated its requirements into its simplest form. Problems of redundancy abounded; many (as high as 200) lower level requirements were then drawn from a single, complex requirement. In effect, this meant that no actual requirement expansion had really taken place since it was difficult to determine, for example, which particular part of a Level 3 requirement was implemented by any given set of Level 4 requirements. However, the project did claim a (pseudo-)decomposition simply because Level 3's were in their own class, Level 4's in theirs and a varying number of links were made between the two classes . In addition, test class structure differed from the requirement class structure to now reflect testing by build. Whereas all the requirements were dumped wholesale into a class and 'requirement expansion' was done across classes using many links, the test classes divided up the test cases first by test type, integration or acceptance, and then by build organization resulting in each test class having minutely defined organizational ownership. This structure required a complex network of linkages between the classes in order to establish the traceability between the different kinds of data (see Figure 13). Complete traceability became difficult. Responsibility for traceability became blurred and changes within a class caused the breaking and re-establishment of linkages.

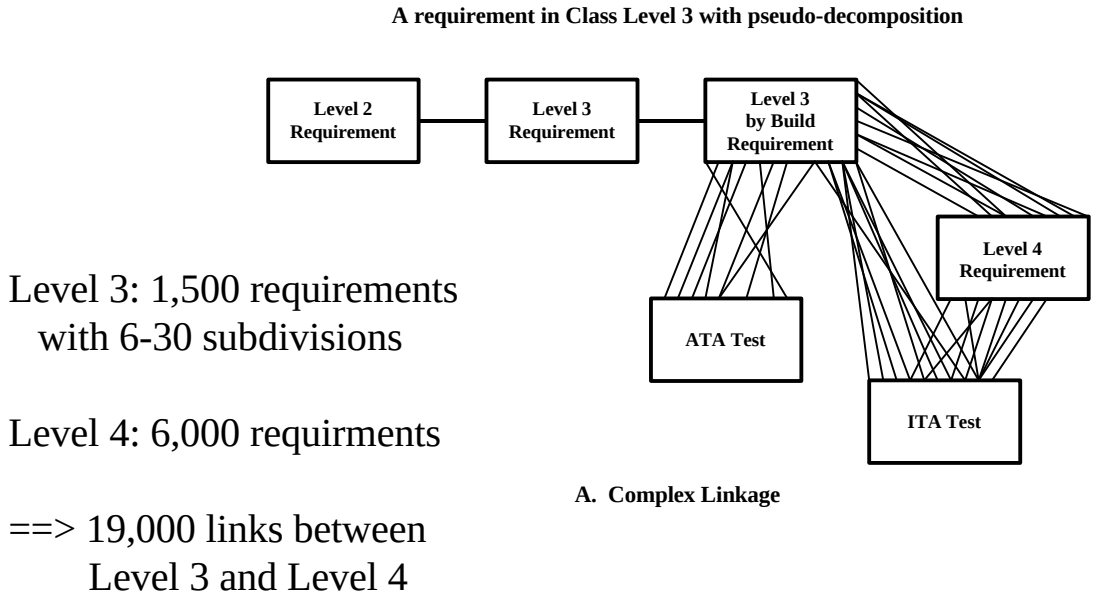


Figure 13 – Project X Complex Schema and Linkages

This also lead to a loss of history associated with changes in the data and important information about the evolution of the system requirements. Lastly, the approach taken to establish a significant number of linkages between individual and complex requirements of the classes lead to degraded performance in the tool. The tool was specifically designed to use a single class for all requirement manipulation at a single level. Inter class links were reserved for infrequently changed relationships. As a result, the project's schema established its main, and highly changeable data with linkages that had high tool processing overhead [6].

In practice, the extremely large nature of this project was not completely understood from the beginning. Had the true size been understood, appreciation for the capabilities of a requirement management tool would have been apparent also. This project averaged 1500 requirements at Level 3 and 6000 requirements at Level 4; the total number of single links between these two classes averaged 19,000 links. However, even these numbers are misleading, as was comprehension of what manipulating this number of requirements truly required. As testimony to this, we found that within each single Level 3 requirement anywhere from 2 to 30 sub-divisions of the requirement might be included. For example: requirement #450 in the database would be counted as one requirement, yet within requirement #450 there is a list of partitions to that requirement which were designated as a, b, c. d, e, f. In reality then, the Level 3 requirement document was describing a minimum of six requirements were only one was counted in the database. At a minimum the project should have decomposed this requirement into the six parts before performing any expansion into the Level 4 requirements. Additionally, where a few links to this one requirement might trace requirement expansion, now 6 times that many links existed between this one large requirement and its next level of implementation [6].

Due to the multiple class approach, links tracing requirements to tests also became extensive and conflicting. Since the project decided to organize the database schema along the lines of the organization, it was necessary to provide the traceability of requirements to requirements and test case to requirement by connections between many classes. The Level 3 by Build requirements were pseudo-decomposed into a set of design requirements at Level 4, as discussed above. This produced unnecessary links and complicated relationships further so there were many more links to each Level 2 and Level 3 requirement than desirable, as stated previously. There was also a stipulation within the project to have traceability between the system test cases (IT) and both Level 3 and Level 4 requirements. This resulted in a complex, undocumentable traceability relationship between the system test cases and the two different levels of requirements. The acceptance test case classes also had undesirable number of links with the high level requirements since it was difficult to understand which part of a test tested which part of a requirement. The tool selected was not designed to efficiently support this kind of usage. Most requirements tools are designed to use minimal classes and effect decomposition within a class and not between classes. The traceability between classes should be used in areas having little change, since the breaking and re-establishment of links between classes is a complex process [6].

This may give the impression that the selection of the tool was the mistake. But it was more than that. This became evident when the project elected to use a new tool. At the time the project was considering the move, it was decided to enlist the aid of a database engineer to assist in the migration of the data to the new tool. It soon became evident that the project in directing how the tool was to be integrated into the project processes was about to make the same mistake again. The database engineer had to explain that the tool capabilities and project management processes must be integrated in such a way as to not emasculate the capabilities of the tool. A thorough understanding is needed of how the tool works and of how the project wants to conduct business. Force fitting one or the other will cause problems to the point where the automated tool no longer can serve the purpose for which it was procured.

6. CONCLUSION

To do requirements right the first time the following component must be present: quality documentation, complete and appropriately structured verification program, and effective requirement management. Quality documentation is complete, clear and concise. This used to be considered ethereal concepts, difficult to measure or visualize. Now with the advent of tools, like ARM, metrics can be developed to see the strengths and weaknesses of the requirement documentation. The completeness of the verification program used to be the only aspect that was easily understood. Now through the use of metrics, a project can gain insight into not only the completeness of the test program but to understand the overall characteristics of the verification program. Effective requirement management now demands the appropriate use of management tools and/or databases through the development life cycle. It is through their use that enables the development of metrics to gain insight to the nature of the requirements. In conclusion, metrics provide projects with a powerful tool to gain insight to each of these areas and give the project the ability to now get the requirements right the first – it is no longer a dream but a reality.

7. REFERENCES

[1] Brooks, Frederick P. Jr., No Silver Bullet: Essence and accidents of software engineering, *IEEE Computer, vol. 15, no. 1,* April 1987, pp. 10-18.

[2] Hammer, T., Huffman, L., Rosenberg, L., Wilson, W., Hyatt, L., "Requirement Metrics for Risk Identification", Software Engineering Laboratory Workshop, GSFC, 12/96.

[3] NASA, *Software Assurance Guidebook,* NASA Goddard Space Flight Center Office of Safety, Reliability, Maintainability, and Quality Assurance, 9/89.

[4] Wilson, W., Rosenberg, L., Hyatt, L., "Automated Analysis of Requirement Specifications", Fourteenth Annual Pacific Northwest Software Quality Conference, 10/96.

[5] Hammer,T., "Measuring Requirement Testing", 18th International Conference on Software Engineering, 5/97.

[6] Hammer,T., "Automated Requirements Management – Beware How You Use Tools", 19th International Conference on Software Engineering, 4/98.

[7] Hansen, Gary W., Hansen, James V., Database Management and Design, Prentice Hall, 1992.

[8] Chen, M., Han, J., Yu, P. "Data Mining: An Overview from a Database Perspective", IEEE Transactions on knowledge and Data Engineering, Vol 8, No. 6, 12/96

8. BIOGRAPHIES

Theodore F. Hammer

Mr. Ted Hammer is the NASA manager for the Software Assurance Technology Center (SATC) at NASA's Goddard Space Flight Center (GSFC). The SATC, through associations with NASA and GSFC projects and organizations, seeks to improve GSFC and NASA software by improving software quality, reducing development risks, and lowering life cycle costs. A prime

focus of the SATC is the provision of software metrics support to GSFC and NASA software development and acquisition projects. In order to meet the needs of these projects, the supporting research done by the SATC is essential to allow the assurance activities to keep pace with the changing software development environment. In addition, the SATC develops techniques, provides software assurance tools, and transfers this technology to NASA and industry.

Prior to this position, Mr. Hammer was a member of the Assurance Management Office where he is responsible for managing the overall quality assurance activities for specific ground system implementation projects, with special emphasis on software quality assurance. Mr. Hammer is also responsible for managing software quality assurance activities for selected spacecraft implementation projects.

Mr. Hammer has over 22 years experience in software development and assurance, 9 with the government at GSFC, and 14 with the government and private industry supporting the Naval Sea Systems Command (NAVSEA). Early in his career he was responsible for test software development for the Combat Direction System on destroyer and frigate classes of ships. He then became responsible for the hardware and software upgrades for the Combat Direction System on these same classes. He moved to private industry, Vitro and ISA, supporting NAVSEA by reviewing software development specifications and witnessing software testing. He later returned to government service (NAVSEA) as project engineer responsible for the implementation, installation and upgrade of the ASW Control System hardware and software on DD963 and AEGIS Class ships. He then worked with the Combat Systems Office and was responsible for planning and coordinating the land based test and evaluation of combat system software upgrades to carriers, cruisers, and destroyers.

He joined NASA/GSFC in 1989. Here he supported NASA Headquarters Software Management Assurance Program, where he participated in the review of the early versions of the military software development standard, MIL-STD-498, as well as NASA software development and assurance standards and guidebooks.

Mr. Hammer received a B.S. in Electrical Engineering from the University of Maryland. He is a member of the American Society for Quality.

Theodore F. Hammer
GSFC
Code 302
Greenbelt, MD 20771
(301) 286-7475 (voice)
(301) 286-1701 (fax)
thammer@pop300.gsfc.nasa.gov

Lenore L. Huffman

Lenore L. Huffman is a principal engineer with the Software Assurance Technology Center (SATC). Ms. Huffman has more than 14 years of software engineering and quality

assurance experience. She is expert in the design, implementation, and execution of data collection, database structures, and metrics reporting and analysis. She is also expert in the design and use of State-Of-The-Art database reporting systems. Ms. Huffman has extensive experience automating Configuration Management and Problem Reporting Systems and adapting their capabilities to satisfy unique project requirements. She has successfully planned, designed, and implemented software quality assurance projects. Prior to joining the SATC, Ms Huffman developed metrics for software at the Space Telescope Institute, and while working at a chemical research center, was awarded with several U.S. patents. Ms Huffman holds a M.B.A. and a B.S.

Lenore L. Huffman
GSFC
Code 300.1, Bld 6
Greenbelt, MD 20771
301-286-0099 (voice)
Lenore.L.Huffman.1@gsfc.nasa.gov

Linda H. Rosenberg, Ph.D.

Dr. Rosenberg is an Engineering Section Head at Unisys Government Systems in Lanham, MD. She is contracted to manage the Software Assurance Technology Center (SATC) through the System Reliability and Safety Office in the Flight Assurance Division at Goddard Space Flight Center, NASA, in Greenbelt, MD. The SATC has four primary responsibilities: Metrics, Standards and Guidance, Assurance tools and techniques, and Outreach programs. Although she oversees all work areas of the SATC, Dr. Rosenberg's area of expertise is metrics. She is responsible for overseeing metric programs to establish a basis for numerical guidelines and standards for software developed at NASA, and to work with project managers to use metrics in the evaluation of the quality of their software. Dr. Rosenberg's work in software metrics outside of NASA includes work with the Joint Logistics Command's efforts to establish a core set of process, product and system metrics with guidelines published in the *Practical Software Measurement*. In addition, Dr. Rosenberg worked with the Software Engineering Institute to develop a risk management course. She is now responsible for risk management training at all NASA centers, and the initiation of software risk management at NASA Goddard. As part of the SATC outreach program, Dr. Rosenberg has presented metrics/quality assurance papers and tutorials at GSFC, and IEEE and ACM local and international conferences. She also reviews for ACM, IEEE and military conferences and journals.

Immediately prior to this assignment, Dr. Rosenberg was an Assistant Professor in the Mathematics/Computer Science Department at Goucher College in Towson, MD. Her responsibilities included the development of upper level computer science courses in accordance with the recommendations of the ACM/IEEE-CS Joint Curriculum Task Force, and the advisor for computer science majors.

Dr. Rosenberg's work has encompassed many areas of Software Engineering. In addition to metrics, she has worked in the areas of hypertext, specification languages, and user interfaces. Dr. Rosenberg holds a Ph.D. in Computer Science from the University of Maryland, an M.E.S. in Computer Science from Loyola College, and a B.S. in Mathematics from Towson State

University. She is a member of Electrical and Electronic Engineers (IEEE), the IEEE Computer Society, the Association for Computing Machinery (ACM) and Upsilon Pi Epsilon.

Dr. Linda Rosenberg
GSFC
Code 300.1, Bld 6
Greenbelt, MD 20771
301-286-0087 (voice)
linda.rosenberg@gsfc.nasa.gov

William M. Wilson

Mr. Wilson is a consultant with the Software Assurance Technology Center (SATC) located at NASA's Goddard Space Flight Center (GSFC). He is also an author and instructor of software safety and reliability courses for System Technology Institute of Malibu, California and a senior associate with Assurance Technology Associates of Mt. Airy, Maryland.

Mr. Wilson has 40 years of professional experience with NASA, DoD, and industry. Prior to joining the SATC, Mr. Wilson was vice-president of Quong and Associates, a San Francisco consulting firm specializing in quality engineering and assurance practices. His area of responsibility in this position was aerospace industry software quality assurance standards and procedures. While Manager of Software Engineering Assurance in the Office of the Chief Engineer at NASA Headquarters, Mr. Wilson established and directed the Software Management and Assurance Program (SMAP). This activity produced NASA's first agency-wide software policies and standards. As a member of the Defense Communications Agency's National Military Command Systems (NMCS) Engineering Directorate, he was the project manager and system engineer for the acquisition and development of several first-generation strategic command, control, and communications systems supporting the National Command Authority .

Mr. Wilson received a BSEE from the Virginia Military Institute. He is a member of the IEEE Computer Society, ACM and ASQC.

William M. Wilson
GSFC
Code 300.1, Bld 6
Greenbelt, MD 20771
301-286-0102 (voice)
wwilson@pop300.gsfc.nasa.gov

Larry Hyatt

Mr. Larry Hyatt is retired from the Systems Reliability and Safety Office at NASA's Goddard Space Flight Center where he was responsible for the development of software implementation policy and requirements. He founded and led the Software Assurance

Technology Center, which is dedicated to making measured improvements in software developed for GSFC and NASA.

Mr. Hyatt has over 35 years experience in software development and assurance, 25 with the government at GSFC and at NOAA. Early in his career, while with IBM Federal Systems Division, he managed the contract support staff that developed science data analysis software for GSFC space scientists. He then moved to GSFC, where he was responsible for the installation and management of the first large scale IBM System 360 at GSFC. At NOAA, he was awarded the Department of Commerce Silver Medal for his management of the development of the science ground system for the first TIROS-N Spacecraft. He then headed the Satellite Service Applications Division, which developed and implemented new uses for meteorological satellite data in weather forecasting. Moving back to NASA/GSFC, Mr. Hyatt developed GSFC's initial programs and policies in software assurance and was active in the development of similar programs for wider agency use. For this he was awarded the NASA Exceptional Service Medal in 1990.

He founded the SATC in 1992 as a center of excellence in software assurance. The SATC carries on a program of research and development in software assurance, develops software assurance guidance and standards, and assists GSFC and NASA software development projects and organizations in improving software processes and products.